

Formal Verification Report – FIFO (synchronous FIFO)

Author: Manish Ranjan

1. Overview

The FIFO design (fifo_sync) implements:

- Synchronous writes and registered reads
- Pointer-based full/empty detection using MSB lap bit
- Occupancy counter 'used'
- Memory addressed by pointer lower bits

Formal verification ensures correctness for all possible input sequences and catches bugs related to ordering, overflow/underflow, and reset behavior.

2. Reset Correctness

Verified:

- FIFO starts empty after reset
- used = 0, empty = 1, full = 0
- Read/write pointers reset to zero

Ensures deterministic startup behavior.

3. Occupancy Counter Correctness

Checked:

- $0 \leq \text{used} \leq \text{DEPTH}$
- $\text{full} \Rightarrow \text{used} == \text{DEPTH}$
- $\text{empty} \Rightarrow \text{used} == 0$
- used changes only by +1, -1, or 0

Prevents overflow, underflow, and count inconsistencies.

4. Pointer Correctness

Verified pointer rules:

- Pointers only increment or hold
- Wrap-around handled correctly
- Pointer distance matches used

Ensures proper circular buffer behavior.

5. Data Integrity (Shadow FIFO Model)

Shadow model tracks:

- Written data sequence
- Expected read order
- Read-before-write behavior in same cycle

Assertion ensures `rd_data` always matches expected.

This proves TRUE FIFO behavior—not just pointer correctness.

6. Polite Environment Assumptions

Assumptions:

- No writes when full
- No reads when empty

Models real-world producer/consumer behavior and prevents unrealistic dead-end situations.

7. Liveness / Progress

Cover statements:

- FIFO can fill to DEPTH
- FIFO can drain to 0

Ensures FIFO is not deadlocked and can make forward progress.

8. Final Result

All properties passed successfully.

FIFO is formally proven:

- Safe
- Data-correct
- Pointer-consistent
- Deadlock-free under polite environment

This makes the FIFO ready for production use.